Lab 04
State Space Search, Search Tree Method, (AND & OR trees)

Objective:

- Understand the concept of State Space Search in Artificial Intelligence.
- Implement an interactive 8-puzzle game.
- Develop a systematic approach to explore possible moves

Activity Outcomes:

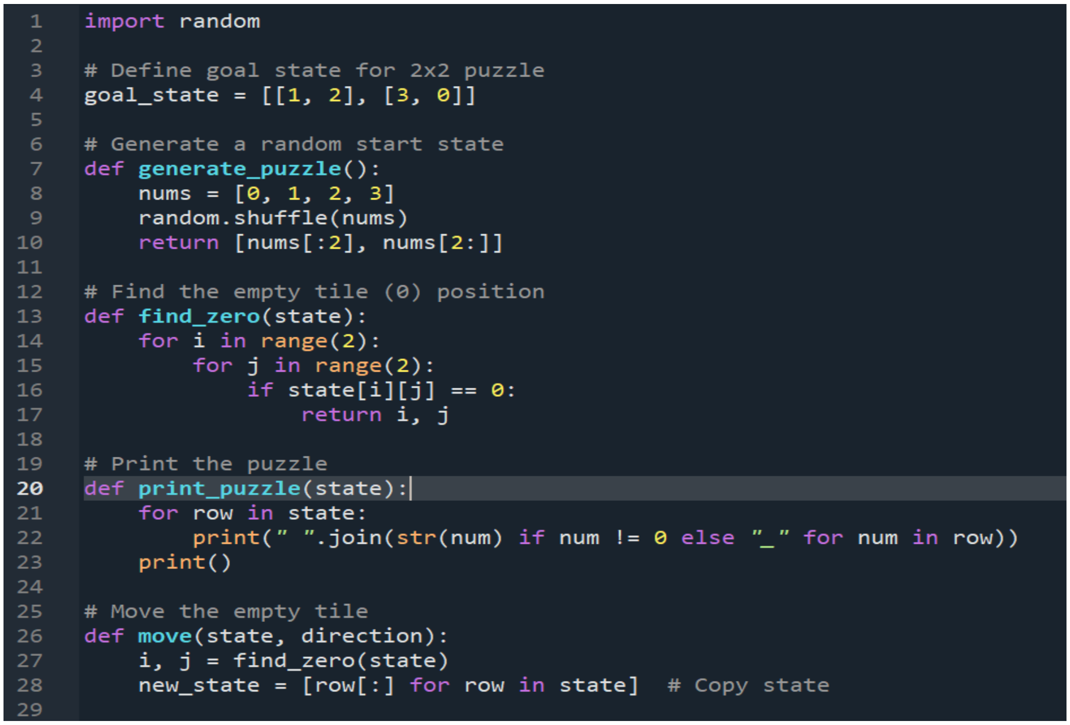
- Students will be able to explain the fundamental concepts of State Space Search.
- Students will demonstrate critical thinking in designing intelligent agent-based solution

1) Useful Concepts

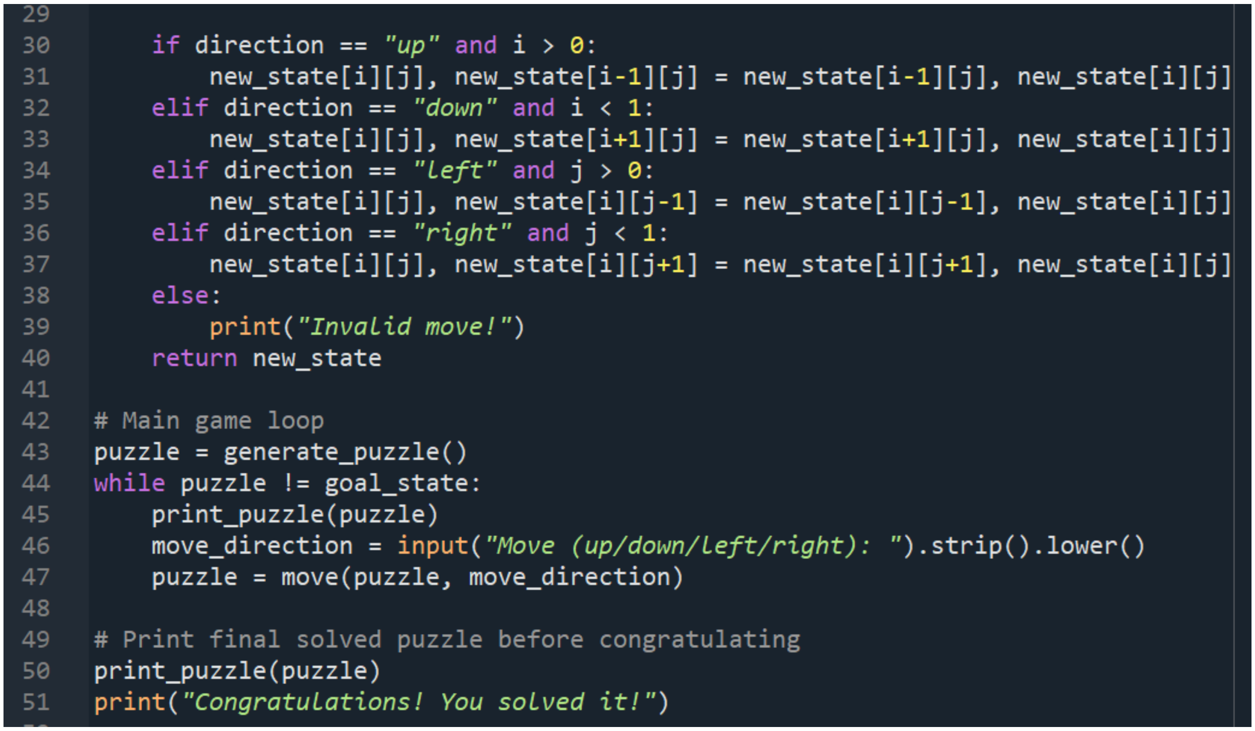
State Space Search is a problem-solving approach in AI where an algorithm explores different possible states to reach a goal. The 8-puzzle problem is a classic example of such a problem.

Activates:

Given a **2x2 sliding puzzle Game** with numbers **1 to 3** and a blank tile (**0**), the goal is to rearrange the tiles to match a predefined **goal state** by moving the blank tile in valid direction



```python
import random

# Define goal state for 2x2 puzzle
goal_state = [[1, 2], [3, 0]]

# Generate a random start state
def generate_puzzle():
    nums = [0, 1, 2, 3]
    random.shuffle(nums)
    return [nums[:2], nums[2:]]

# Find the empty tile (0) position
def find_zero(state):
    for i in range(2):
        for j in range(2):
            if state[i][j] == 0:
                return i, j

# Print the puzzle
def print_puzzle(state):
    for row in state:
        print(" ".join(str(num) if num != 0 else "_" for num in row))
    print()

# Move the empty tile
def move(state, direction):
    i, j = find_zero(state)
    new_state = [row[:] for row in state]  # Copy state

```



```
    if direction == "up" and i > 0:
        new_state[i][j], new_state[i-1][j] = new_state[i-1][j], new_state[i][j]
    elif direction == "down" and i < 1:
        new_state[i][j], new_state[i+1][j] = new_state[i+1][j], new_state[i][j]
    elif direction == "left" and j > 0:
        new_state[i][j], new_state[i][j-1] = new_state[i][j-1], new_state[i][j]
    elif direction == "right" and j < 1:
        new_state[i][j], new_state[i][j+1] = new_state[i][j+1], new_state[i][j]
    else:
        print("Invalid move!")
    return new_state

# Main game loop
puzzle = generate_puzzle()
while puzzle != goal_state:
    print_puzzle(puzzle)
    move_direction = input("Move (up/down/left/right): ").strip().lower()
    puzzle = move(puzzle, move_direction)

# Print final solved puzzle before congratulating
print_puzzle(puzzle)
print("Congratulations! You solved it!")
```

Search Tree Method

A search tree is a structured way to explore possible states or paths in a problem-solving scenario. It is commonly used in Artificial Intelligence (AI) and pathfinding problems to represent decision-making processes.

Key Components of a Search Tree:

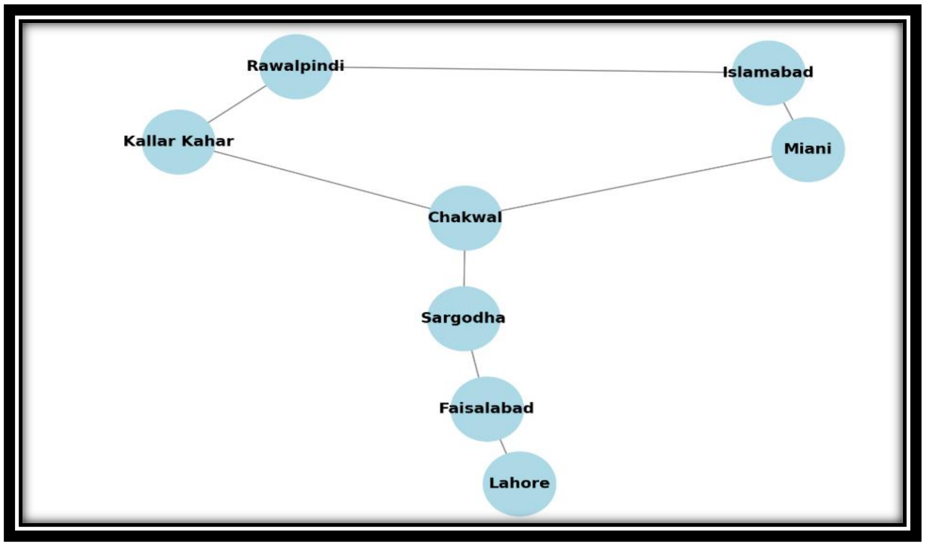
- **Root Node** → Represents the starting state (e.g., the initial city in the pathfinder game).
- **Branches (Edges)** → Connections between states (e.g., roads between cities).
- **Child Nodes** → Possible states that can be reached from the current state.
- **Goal Node** → The desired final state (e.g., reaching Lahore from Islamabad).

How It Works:

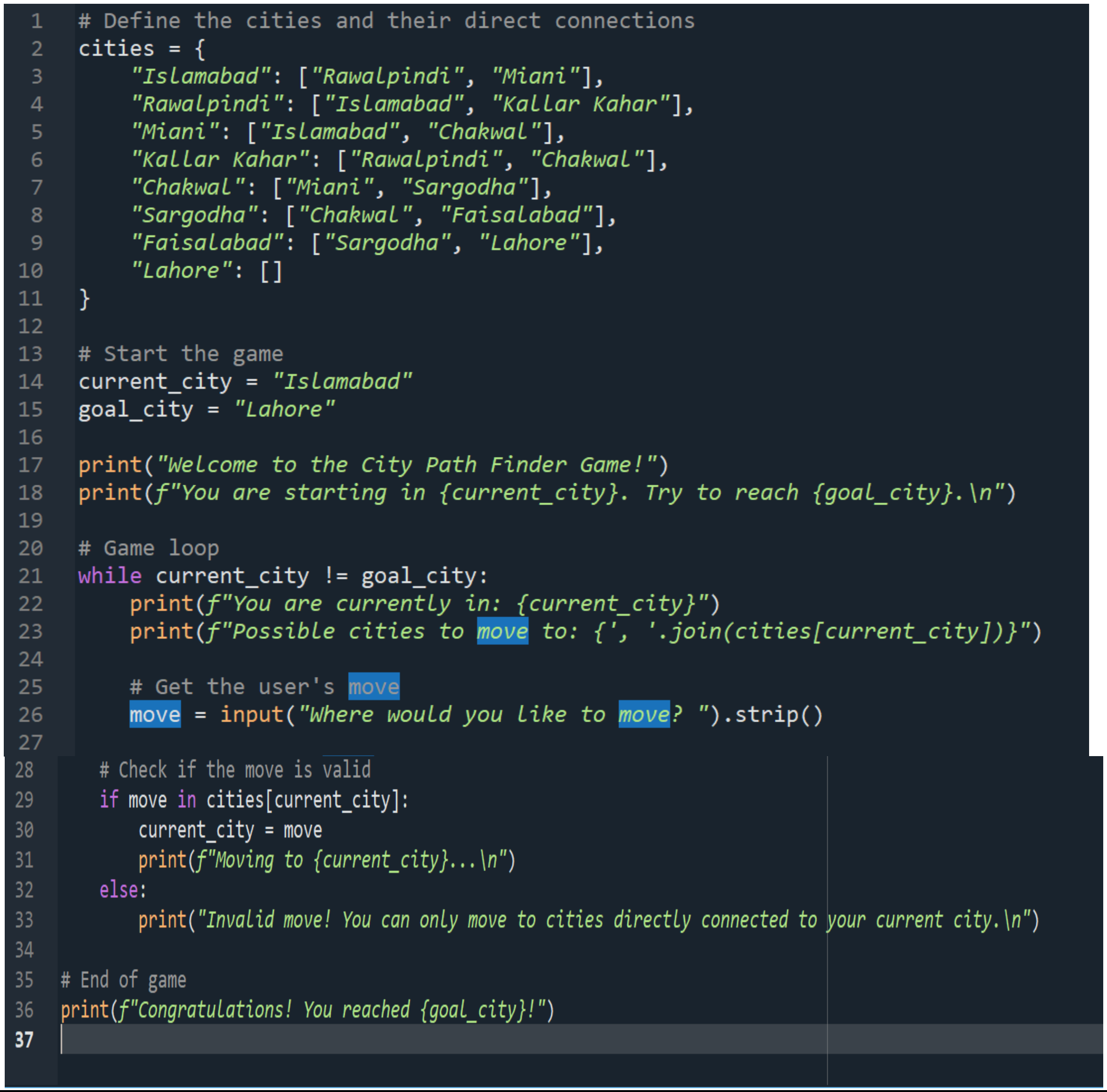
- The tree starts from the root node (initial state).
- Each node expands into possible next states, forming branches.
- The search continues until it reaches the goal state.

2) Lab Activity

Path Finder Game:



Code:



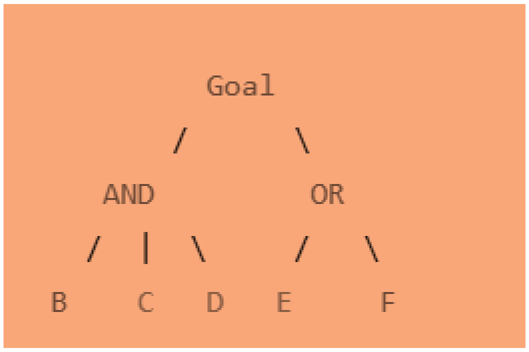
```python
# Define the cities and their direct connections
cities = {
    "Islamabad": ["Rawalpindi", "Miani"],
    "Rawalpindi": ["Islamabad", "Kallar Kahar"],
    "Miani": ["Islamabad", "Chakwal"],
    "Kallar Kahar": ["Rawalpindi", "Chakwal"],
    "Chakwal": ["Miani", "Sargodha"],
    "Sargodha": ["Chakwal", "Faisalabad"],
    "Faisalabad": ["Sargodha", "Lahore"],
    "Lahore": []
}

# Start the game
current_city = "Islamabad"
goal_city = "Lahore"

print("Welcome to the City Path Finder Game!")
print(f"You are starting in {current_city}. Try to reach {goal_city}.\n")

# Game loop
while current_city != goal_city:
    print(f"You are currently in: {current_city}")
    print(f"Possible cities to move to: {', '.join(cities[current_city])}")

    # Get the user's move
    move = input("Where would you like to move? ").strip()

    # Check if the move is valid
    if move in cities[current_city]:
        current_city = move
        print(f"Moving to {current_city}...\n")
    else:
        print("Invalid move! You can only move to cities directly connected to your current city.\n")

# End of game
print(f"Congratulations! You reached {goal_city}!")
```

AND & OR Tree

An AND-OR Tree is a decision-making structure used in problem-solving, especially in AI, to handle multiple possible solutions and conditional dependencies.

AND-OR Tree Combination

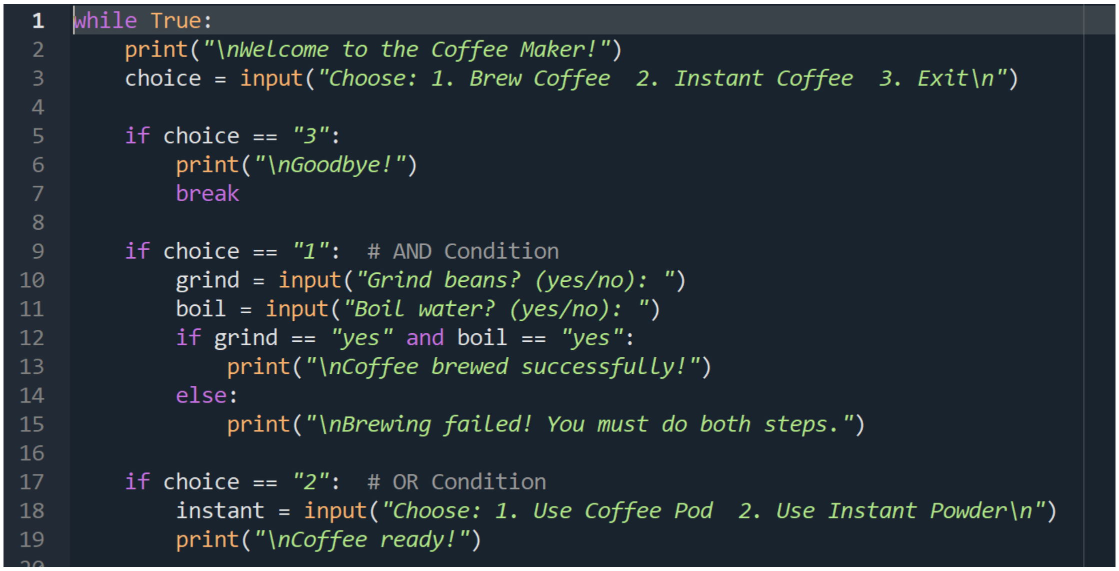
- Some problems have both AND and OR conditions.
- Example: A game where you need either a key OR a password to open a door, AND you must also defeat a guard.



Use Case:

- Completing a mission where some tasks are mandatory (AND) while others offer multiple choices (OR).

Make Coffee Maker with AND and OR Trees



```python
while True:
    print("\nWelcome to the Coffee Maker!")
    choice = input("Choose: 1. Brew Coffee  2. Instant Coffee  3. Exit\n")

    if choice == "3":
        print("\nGoodbye!")
        break

    if choice == "1":  # AND Condition
        grind = input("Grind beans? (yes/no): ")
        boil = input("Boil water? (yes/no): ")
        if grind == "yes" and boil == "yes":
            print("\nCoffee brewed successfully!")
        else:
            print("\nBrewing failed! You must do both steps.")

    if choice == "2":  # OR Condition
        instant = input("Choose: 1. Use Coffee Pod  2. Use Instant Powder\n")
        print("\nCoffee ready!")
```

3) Graded Lab Tasks (Allotted Time 1.5 Hours)

1.

Given a **3x3 sliding puzzle Game** with numbers **1 to 8** and a blank tile (**0**), the goal is to rearrange the tiles to match a predefined **goal state** by moving the blank tile in valid direction

2.

Create Path Finder game in which destination from Murree to Karachi.

Hint:

```
"Murree": ["Islamabad"],
    "Islamabad": ["Murree", "Rawalpindi"],
    "Rawalpindi": ["Islamabad", "Kallar Kahar"],
    "Kallar Kahar": ["Rawalpindi", "Chakwal"],
    "Chakwal": ["Kallar Kahar", "Sargodha"],
    "Sargodha": ["Chakwal", "Faisalabad"],
    "Faisalabad": ["Sargodha", "Lahore"],
    "Lahore": ["Faisalabad", "Multan"],
    "Multan": ["Lahore", "Sukkur"],
    "Sukkur": ["Multan", "Hyderabad"],
    "Hyderabad": ["Sukkur", "Karachi"],
    "Karachi": []
```

3.

Python code simulating **AND Gate (Office Door Access)** and **OR Gate (Street Lights Control)** with real-world scenarios